Preprint: full publication in Michael Blackstock and Rodger Lea. 2016. FRED: A Hosted Data Flow Platform for the IoT. In Proceedings of the 1st International Workshop on Mashups of Things and APIs (MOTA '16). ACM, New York, NY, USA, Article 2, 5 pages. DOI: https://doi.org/10.1145/3007203.3007214

FRED: A Hosted Data Flow Platform for the IoT built using Node-RED

Michael Blackstock
Sense Tecnic Systems, Inc.
308 E. 5th Ave.,
Vancouver, Canada
mblackstock@sensetecnic.com

Rodger Lea
School of Computing & Comms,
Lancaster University, UK
also with University of British Columbia, Canada
rodger@comp.lancs.ac.uk

ABSTRACT
IoT developers need to integrate a variety of protocols, backend components and services; they often need to pre and post-process data as well as react to changes in the real world. Data flow programming tools have been introduced in a number of related domains to provide a flexible, but easy to use visual programming environment for rapid development. The open source Node-RED system provides such a tool for IoT applications, but is limited to executing a single flow file in a single thread. In this paper we describe the design of our system called the Front-End for Node-RED (FRED) that manages multiple instances of Node-RED for logged in users, allowing Node-RED to be used as a cloud-hosted data flow mashup tool for the IoT. We present some examples of how some of our 1800+ users are using FRED for IoT mashups, and some of the challenged we faced in implementing the FRED system.

CCS Concepts
• Information systems → World Wide Web → Web Interfaces → Mashups • Software and its engineering → Software notations and tools → General programming languages → Language types → Data flow languages.

Keywords
Internet of Things; IoT Mashups; Node-RED; Data Flow Programming

1. INTRODUCTION
Building IoT applications can be a complex task. It requires the integration of a variety of protocols and services, as well as a secure and flexible backend to manage and gather volumes of time series data streams from 'things' and services. Developers need a means to pre and post-process data and mechanisms to alert and react to changes in both the physical and online world. While it is possible to create real time interactive IoT mashups using traditional programming tools, it can be difficult, requiring developers to learn new protocols and APIs, create data processing components, and link them together. To provide more flexibility while maintaining ease of use, several systems provide a data-flow programming paradigm [12] where computer programs are modeled as directed graphs connecting networks of 'black box' nodes that exchange data along connected arcs. Visual data flow programming languages have been used in many other domains such as high performance parallel computing [1] leveraging multi-core processors, music [17], toys, science and engineering [20].

In 2010 our group created the WoTKit, an IoT platform which included a cloud-hosted visual dataflow programming system called the Processor for creating IoT mashups [2]. Using the tool, developers created *pipes* made-up of input, data processing and output modules acting on data received in real time as sensors and other data sources pushed data into the WoTKit platform. In 2014, our group began using the Node-RED open source project developed by IBM. The initial focus for Node-RED was edge devices, offering a visual data flow tool to address the issues around device-level integration by providing a variety of pre-built components for integration and data processing, using a visual data flow paradigm similar to that used by the cloud-hosted Processor and other visual data flow programming languages.

This is a pre-print version of a paper presented at the 1st Workshop on the Mashup of things and APIs (MOTA) part of the 2016 ACM Middleware workshop in Trento, Italy.

The final version is available in the ACM digital library.

http://dl.acm.org/citation.cfm?id=3007214

What is most impressive about Node-RED is the large and growing community of developers and variety of nodes they contribute for others to use in their IoT projects. They range from nodes for making it easy to connect to the GPIO pins on a Raspberry Pi to connecting to on-line notification, instant messaging systems and social networks. We decided to refocus our efforts on the Processor to leverage the large community and growing ecosystem of nodes available to Node-RED. By doing so, we believed this would have the added benefit of sharing the same data-flow programming model on both devices and the cloud. To get started we embarked on a comparison and evaluation of the Processor and Node-RED systems [3, 9]. This effort

was followed by the design and implementation of the system we call FRED: *a Front-end for Node-RED*. FRED provides a multi-tenant cloud hosted Node-RED system for rapid cloud-hosted IoT integration and application development.

While it is possible to host Node-RED on cloud infrastructures such as Amazon Web Services or platforms such as the Cloud Foundry-based BlueMix [6, 10], with FRED, we aim to make it as fast and easy to get started with Node-RED as possible. Today, FRED requires no subscriptions, set up, installation or configuration for users to begin creating cloud-hosted mashups using Node-RED. They simply register on the system and hit 'Start Instance'. This makes it faster and easier for users to try out Node-RED for rapid IoT and service mashups.

In the remainder of this paper we provide a short overview of Node-RED, then describe the requirements and design of FRED. This is followed by some examples of FRED-hosted flows, a discussion of our experience to date with our FRED deployment, and some hints at future work. By participating in the workshop we would like to demonstrate FRED as a tool for rapid prototyping of applications, get feedback on our system to date and discuss open research issues around cloud-hosted Node-RED and similar tools.

2. NODE-RED

Node-RED is a web-based tool for 'wiring up' hardware devices with various protocols and APIs. It provides both a browser-based visual editor and a run time implemented in JavaScript using the Node.js framework. By doing so, it takes advantage of the Node.js built in asynchronous event-driven runtime and native support for JavaScript on both the browser and on the server.

Programs written using Node-RED are called *flows*. They consist of a set of nodes connected by wires (see Figure 4). Like other visual data flow systems, the user interface consists of a visual flow editor where node templates, representing different input, output and data processing nodes, can be dropped into the canvas and wired together.

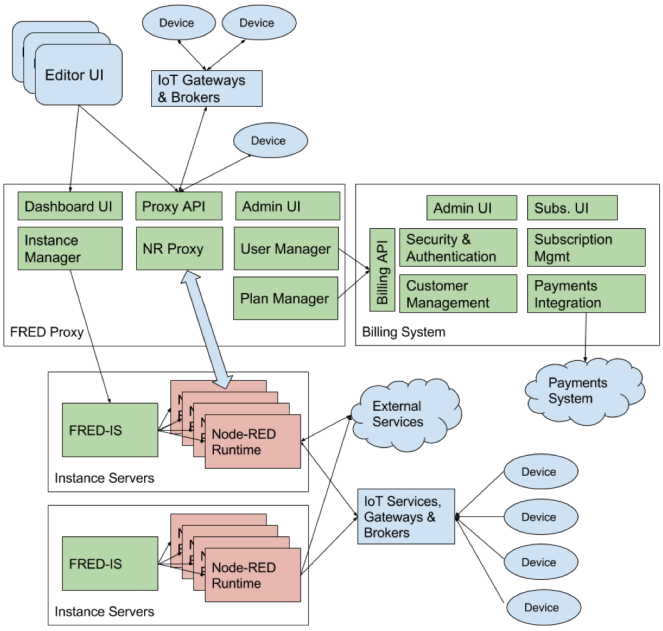

Figure 1. FRED High Level Architecture

To execute a flow, the flow file is read, and nodes are instantiated that correspond to the node type in the flow file. Nodes and node sets are usually packaged and installed as node.js modules. The nodes in a set inherit from a Node base class, a subclass of an *EventEmitter* in the Node.js event API. The Node object implements the observer design pattern to maintain subscriber lists defined by wires, and emits events to nodes downstream in a flow.

On instantiation, input nodes may subscribe to external services, begin listening for data on a port, get data from a local sensor or file, or begin processing HTTP requests for example. Once data is processed by a node, either from an external service, or received from an upstream node via its "input" handler, a node calls the base class Node send() method with a JavaScript object to send named events or messages to the single threaded Node.js event loop for execution. The event loop processes these messages using the downstream Node instances that can process data, generate additional events, or communicate with outside services, local hardware or the OS.

It is possible to store global, per tab and per-node state using Node-RED in what are called *context variables*. These variables persist between executions of the flow. Because Node-RED uses the single-threaded Node.js runtime it cannot easily take advantage of multiple cores on a server.

3. FRED DESIGN

The Front-End for Node-RED is a service that hosts multiple Node-RED processes, currently one instance for each registered user. The FRED system is essentially a 'smart' proxy that creates and manages Node-RED processes, and 'proxies' communications between connected clients, devices, and services. Like Node-RED itself, it is implemented using Node.js.

The FRED service can be used to connect devices to cloud services, coordinate communication between devices, integrate services with each other or for creating new web APIs and applications. To do this, it supports the ability to run flows for multiple users; all flows should get fair access to CPU, memory and storage resources. It provides secure access to flow editors and the flow run-time. The system scales with the number of users and their hosted flows.

In the design of the FRED system we aimed to take advantage of the open source community's work on both the Node-RED platform and nodes as they evolve. To do this, we aimed to minimize the changes to both while addressing other constraints. While Node-RED is delivered as a web service, it has a number of limitations. In particular, it was designed for hosting one 'flow'[1] on one device/host; at a minimum we needed to find a way to deploy Node-RED in the cloud for many users and their flows efficiently and cost-effectively. We need to have a way to monitor Node-RED instances and restart them automatically in case it crashes or exhibits node configuration problems.

[1] While Node-RED displays multiple data flow segments in different tabs, they are effectively one single flow executed in one thread by the Node-RED execution engine.

The FRED system consists of three sub-systems as shown in Figure 1. FRED-IS (Instance Server) which manages individual instances, the FRED Proxy which provides a management front end for users and authorized administrators and the FRED billing service.

The FRED Proxy is used to create, start and stop a Node-RED process, delegating the work of managing these processes to FRED-IS (described next). As its name implies, FRED Proxy includes a proxy component that redirects communication from each user's browser to their node-red process, devices and services that are communicating with a Node-RED flow using HTTP or web sockets. Depending on the type of client (browser, device or service) this is done using a browser cookie, API key for devices to access the system, or URL-based mapping for public access to nodes. The FRED Proxy provides the administrative UI described earlier for managing users and Node-RED processes.

Instance Servers host FRED-IS services. These FRED-IS micro services manage the node-red processes running on the host. To provide isolation from the OS and limit the memory and CPU, FRED-IS leverages the Docker container system [7], using the Docker API to manage processes on the host.

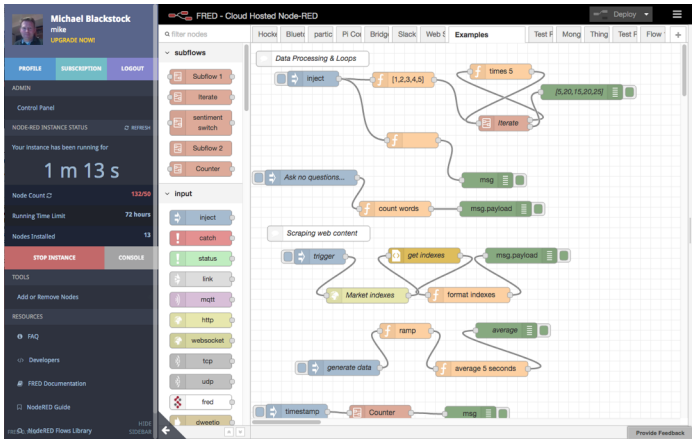

Figure 2. FRED Extensions to Node-RED Editor

Finally, the FRED billing service provides authentication and plan subscription services. Once registered and logged in, the Billing System creates a JSON Web Token that is checked by FRED to ensure they are authorized to use the system.

To provide redundancy and scalability the FRED Proxy and Billing System are replicated behind load balancers. Additional Instance Servers are added to a cluster to support additional instances as demand for the number of concurrently running Node-RED processes increases.

3.1 User Interface

The FRED UI extends the Node-RED UI by adding a management panel to the left of the standard Node-RED editor interface called the Dashboard as shown in Figure 2. This panel provides controls for the user to start and stop the Node-RED process, and view the standard output and error streams generated by node red, provides access to their user profile and subscription related information. Access to standard output and error streams is needed since some information needed by developers such as status output from the Node-RED execution engine and node themselves is only shown there.

3.2 Administration

Administrators can manage user profile information as well as log into the system as a given user to provide support and alter a user's flows on their behalf.

Using the instance panel, an administrator can start and stop instances, and configure *limits*: the limitations on an instance such as maximum memory used, node count and time to run that are mapped to subscription plans. For example, with a free subscription, instances run for only 72 hours between log-ins, and are limited to 50 node whereas paid subscriptions run longer, and have higher node limits.

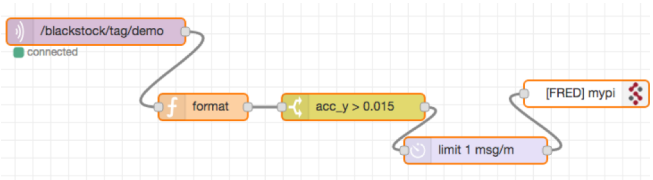

Figure 3. Flow to receive data from sensor tag, and send alert to a Raspberry Pi

4. EXPERIENCE

Currently FRED is deployed on Amazon AWS and is available for free public access[2]. There are approximately 1900 registered users of the system to date with users ranging from those just learning Node-RED to home automation hobbyists through to companies using FRED for industrial IoT applications such as factory automation, precision agriculture, environmental monitoring and process control. Cutting across those use domains are a set of different use patterns, ranging from classic IoT device/sensor monitoring and alerting, through to simple web services and bots up-to back-end integration of enterprise software services.

In support of this diverse set of use patterns, FRED includes a variety of additional nodes for connectivity to external services such as databases, social networks, messaging, notification services and hosted enterprise services.

4.1 Example Flows

To illustrate the diversity of use cases for a cloud-hosted Node-RED system like FRED, we present two example flows. Many FRED users connect devices to FRED through gateways that use a protocol suitable for wide area networks such as HTTP, web sockets or MQTT. To illustrate, we'll connect a TI Sensor tag to the cloud. This device is a sensor hardware package that collects data from a number of onboard sensors including accelerometer, buttons humidity and temperature, and sends them to a mobile phone application via Bluetooth Low Energy protocol [4]. We configured the TI Sensor tag application to send data to a public MQTT server at test.mosquitto.org.

[2] FRED service is at https://fred.sensetecnic.com

With this device and gateway configuration, it is straightforward to create a flow to retrieve this data and use it for data processing in the cloud. For example, in Figure 3 we have a simple flow that will send an 'alarm' message when the Y accelerometer data from the sensor meets a condition. An MQTT input node receives data from the device connected to a smart phone. The *function* node called "format" to makes changes to the JSON message fields and types received from the SensorTag. The *switch* node passes messages only when the acc_y payload value exceeds 0.015. Finally, a *delay* node ensures no more than one message per minute is sent to a FRED output node, used to signal an alarm sound on a Raspberry Pi connected to a wifi router.

FRED is used by some users to develop and host RESTful APIs and even simple web sites. The flow shown in Figure 4.b implements a simple real time chat application shown in Figure 4.a using public HTTP and web sockets endpoints.

The endpoint */public/chat* is a public HTTP node that serves a web page containing the chat UI using a template node. This application communicates with the system through the */public/receive* and */public/publish* web sockets nodes to send and receive data to other users. When data is received it is run through the same sentiment node so that the application can colour the message green (positive), blue (neutral or unknown) or red (negative) in the chat window.

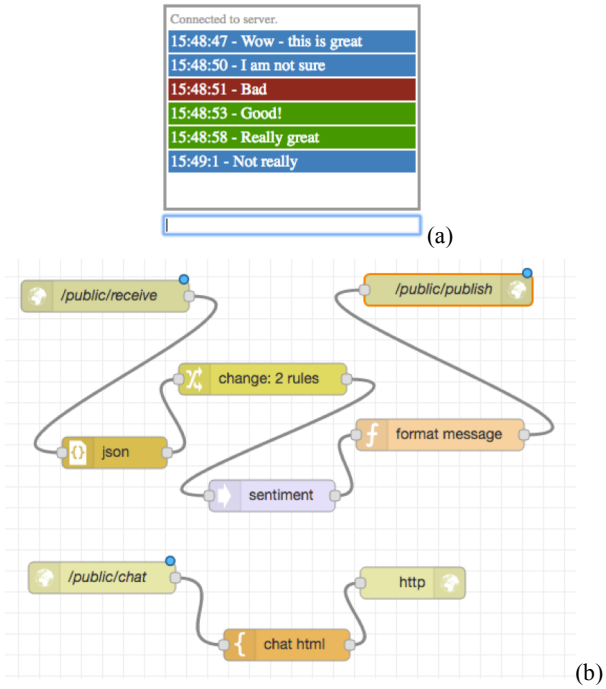

Figure 4 (a) Real time chat application user interface. (b) Simple flow that implements real time chat with sentiment analysis.

Table 1. How would you describe your use of FRED

Question	Number	Percent
To learn about Node-RED	328	62.8%
For personal projects	129	24.7%
For work – but just playing around	25	4.8%
For a company project	12	2.3%
For my own business (consultant, contractor)	8	1.5%
Other	20	3.8%

Table 2. What industry or area of interest are you using FRED for?

Question	Number
Home Automation	294
Remote Monitoring and Notifications	231
Web services and APIs	176
Backend integration	95
Process control/factory automation	94
As part of a hardware dev kit	90
Bots/Chat applications	61
Other	101

4.2 Understanding FRED usage

To understand what users are doing with FRED, we surveyed 522 FRED users when they registered themselves with the system asking *How would you describe your use of FRED*, and *What Industry or area of interest are you using FRED for (check all that apply)*. The responses are summarized in Tables 1 and 2.

Based on this survey, today FRED is being used mostly as a tool to learn about Node-RED, and for personal projects. The use cases align with this, with most people using FRED for home automation and monitoring. It is interesting to note that some users see FRED as a useful tool for creating web services APIs.

5. DISCUSSION

FRED in its current implementation has a few limitations mostly related to the decision to use Node-RED in its current form, which leverages the Node.js runtime. Other limitations are related to hosting node-red as a service, and how FRED proxies interaction to node-red instances by transforming URLs.

Today, users can create their own function nodes and sub flows, but FRED currently does not allow users to install their own nodes. This is because many nodes make assumptions about the Node-RED host including access to the underlying OS. Since nodes have access to the host and the node-red runtime, it is easy for a user to get into a situation where their instance will not start. Until we have implemented easy ways for users to fix these issues, we only allow users to use certain pre-tested nodes on FRED.

The Node.js runtime has a fairly large memory footprint. When we run several instances of Node-RED we duplicate all of the Node.js libraries in RAM; there is no easy way to share this code. This is exacerbated when many users install the same nodes; each of these uses

more RAM on a cloud instance, which is duplicated for every user that makes use of them. There is not a clear way to address this issue without changes to the Node.js and Node-RED run time. To keep RAM use to a minimum, we recently added a feature to allow users to manage the nodes installed in their instance. This conserves RAM since many users will only use a small subset of available nodes. Users select from a list of pre-qualified nodes, and restart their Node-RED instance for the changes to take effect.

Several node-red nodes assume that the user has access to the host and the underlying file system. Since we are providing node-red as a service, and not complete access to the process and host, this means that a number of nodes such as the file I/O nodes and Sqlite nodes that require local file access cannot be included without changes. The freeboard dashboard node saves dashboard information to the local file system. If the user does not remember to bookmark their saved dashboard, there is no easy way for them to retrieve them. To address this we aim to provide a file browser/editor for users to access their file storage directory and limit the file size of files used by certain nodes such as sqlite.

As a mashup tool, FRED has been found useful as a quick and easy way to try out Node-RED without the need to install node.js and Node-RED on their laptops or devices. It has been used in several IoT hackathons and as a teaching tool aimed at students and non-technical users. One non-profit organization[3] is using FRED to teach kids how to use LoRaWan sensor networks to collect, process and visualize environmental sensor data. Many active users are running simple flows to monitor and manage their homes.

Non-technical users of FRED can produce flows that are not efficient, or not well organized. In some cases, these flows may use more network or CPU resources than is necessary. One user created many MQTT server configurations to the same MQTT broker to avoid connectivity problems. Others have accidentally introduced infinite loops or allocate many objects in a function node, causing their Node-RED instance to deadlock or run out of RAM. When these issues occur, we have needed to provide direct support to improve or fix flows. While FRED and Node-RED provides a lot of flexibility, for non-developers, it is also possible to get into trouble.

6. RELATED WORK

There are several IoT platforms that include a visual flow-based editor similar to the Node-RED system hosted by FRED. Zhang et al. [21] use an HTML-based visual sensor logic design to create 'virtual sensors' that are based on data from physical sensors that are part of a standards-based sensor network. On a smaller scale, L Mainetti et al. built a mash up platform based on the ClickScript visual programming language [5] that connects devices on constrained networks using the COAP protocol [15]. Meyer et al. integrated their semantic smart environments system with ClickScript to allow end users to graphically model their smart environments [18].

If This Then That (IFTTT) [11] allows users to create condition-action triggers to connect cloud services, including IoT backend services to each other. An IFTTT recipe can be configured to turn on the lights in your home when the alarm system detects an intruder. Unlike Node-RED's data flow run time, the IFTTT system exposes a pre-configured set of events and actions that are possible with a given pair of service connectors.

Like Node-RED, NoFlo [19] is a flow based programming environment for Javascript. It was not targeted to IoT applications, and instead to ease the development of web applications. Like Node-RED, NoFlo provides a visual editor for a data flow runtime leveraging Node.js. Unlike FRED, NoFlo is not a cloud service; developers install and deploy NoFlo themselves, on it's own, or embedded in an application. GlueThings Composer is a fork of Node-RED designed to connect to other components of the GlueThings platform: the Device Manager, and Deployment Manager [13].

7. CONCLUSIONS

To date FRED has proven to be a useful tool for users learning about Node-RED and rapidly prototyping cloud-hosted applications. To make the tool more useful we intend to take advantage of enhancements to Node-RED itself, and adding features to FRED. This includes adding support for version control of flows, adding support for developer collaboration and allow users to edit their flows and other files using a text editor. We will allow users to manage more than one instance of Node-RED and share instances with other FRED users.

For non-developers we are considering a 'simple mode' that reduces some of the flexibility offered by Node-RED in favour of addressing certain common mash up use cases such as home monitoring quickly and easily. This may include removing some core node such as the function node, and providing different sets of example flows on startup.

Longer term, we will explore the use of Docker Swarm [8], Kubernetes [14], Marathon [16], and Cloud Foundry, to simplify the multi-host architecture and make it easy for users to transition from a prototype container to production deployments of Node-RED. We anticipate creating nodes that represent data processing services that do not run within the Node-RED runtime. Finally, we aim to make it easier to manage and distribute flows hosted both in the FRED cloud service and on devices [3].

8. ACKNOWLEDGMENTS

Our thanks to Nick O'Leary and Dave Conway-Jones at IBM and others in the Node-RED community for the Node-RED system, nodes and feedback in the development of FRED, and to Ted Huang at Sense Tecnic who helped design and implement FRED.

9. REFERENCES

[1] Ackerman, W.B. 1982. Data Flow Languages. *Computer*. 15, 2 (Feb. 1982), 15–25.

[2] Blackstock, M. and Lea, R. 2012. IoT mashups with the WoTKit. *Internet of Things (IOT), 2012 3rd International Conference on the* (Wuxi, China, Oct. 2012), 159–166.

[3] Blackstock, M. and Lea, R. 2014. Toward a Distributed Data Flow Platform for the Web of Things (Distributed Node-RED). *Proceedings of the 5th International Workshop on Web of Things* (New York, NY, USA, 2014), 34–39.

[3] https://herelab.io/stem-camps/ HereLab STEM Camps.

[4] Bluetooth Low Energy | Bluetooth Technology Website: *https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics/low-energy*. Accessed: 2016-08-26.
[5] ClickScript-Server: *http://clickscript.ch/site/home.php*. Accessed: 2016-08-22.
[6] Cloud Foundry | The Industry Standard for Cloud Applications: *https://www.cloudfoundry.org/*. Accessed: 2016-10-12.
[7] Docker: *https://www.docker.com/*. Accessed: 2016-08-26.
[8] Docker Swarm: 2016. *https://docs.docker.com/swarm/*. Accessed: 2016-10-12.
[9] Giang, N.K. et al. 2015. Developing IoT applications in the Fog: A Distributed Dataflow approach. *Internet of Things (IOT), 2015 5th International Conference on the* (Oct. 2015), 155–162.
[10] IBM Bluemix - Next-Generation Cloud App Development Platform: *https://console.ng.bluemix.net/*. Accessed: 2016-10-12.
[11] IFTTT / Put the internet to work for you.: *https://ifttt.com/*. Accessed: 2016-08-22.
[12] Johnston, W.M. et al. 2004. Advances in Dataflow Programming Languages. *ACM Comput. Surv.* 36, 1 (Mar. 2004), 1–34.
[13] Kleinfeld, R. et al. 2014. Glue.Things: A Mashup Platform for Wiring the Internet of Things with the Internet of Services. *Proceedings of the 5th International Workshop on Web of Things* (New York, NY, USA, 2014), 16–21.
[14] Kubernetes - Production-Grade Container Orchestration: *http://kubernetes.io/*. Accessed: 2016-10-12.
[15] Mainetti, L. et al. 2013. A novel architecture enabling the visual implementation of web of Things applications. *2013 21st International Conference on Software, Telecommunications and Computer Networks (SoftCOM)* (Sep. 2013), 1–7.
[16] Marathon: A container orchestration platform for Mesos and DCOS: *https://mesosphere.github.io/marathon/*. Accessed: 2016-10-12.
[17] MAX is a visual programming language for media: *http://cycling74.com/products/max/*. Accessed: 2016-08-22.
[18] Mayer, S. et al. 2014. Configuration of smart environments made simple: Combining visual modeling with semantic metadata and reasoning. *Internet of Things (IOT), 2014 International Conference on the* (Oct. 2014), 61–66.
[19] NoFlo: Flow-Based Programming for JavaScript: *http://noflojs.org/*. Accessed: 2016-08-22.
[20] What is LabVIEW: 2013. *http://www.ni.com/newsletter/51141/en/*. Accessed: 2016-08-22.
[21] Zhang, J. et al. 2013. Supporting Personizable Virtual Internet of Things. *Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on Autonomic and Trusted Computing (UIC/ATC)* (Dec. 2013), 329–336.